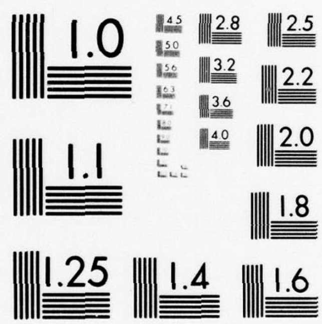
AD-A062 753 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 5/2
DATA MODEL EQUIVALENCE,(U)
DEC 78 S A BORKIN N00014-75-C-0661
UNCLASSIFIED MIT/LCS/TM-118 NL

1 OF 1
AD
A062753

END
DATE
FILMED
3-79
DDC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

LEVEL II

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

(12)

MIT/LCS/TM-118

DATA MODEL EQUIVALENCE

Sheldon A. Borkin

December 1978

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract N00014-75-C-0661

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(14)

1. REPORT NUMBER MIT/LCS/TM-118	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) Data Model Equivalence		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER MIT/LCS/TM-118
7. AUTHOR(s) (10) Sheldon A. Borkin		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-75-C-0661
9. PERFORMING ORGANIZATION NAME AND ADDRESS MIT/Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139 409 648		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (11) Dec 78
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency Department of Defense 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE November 1978 13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Department of the Navy Information Systems Program Arlington, VA 22217 (12) 38p.		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

database systems
semantic data models
data model equivalence
semantic relation data model
semantic graph data model

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The current proliferation of proposals for database system data models and the desire for database systems which support several different models raise many questions concerning "equivalence properties" of different data models. To answer these questions, one first needs clear definitions of the concepts under discussion. This paper presents formal definitions of the terms database, operation, operation type, application model and data model.

Using this formal framework, database state equivalence, operation

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 648

LB

20. equivalence, application model equivalence and data model equivalence are distinguished. Three types of application and data model equivalence are defined - isomorphic, compised operation and state dependent. Possibilities for partial equivalences are mentioned. Implementation implications of these different equivalences are discussed.

Examples are presented using two semantic data models, the semantic relation data model and the semantic graph data model.

MIT/LCS/TM-118

DATA MODEL EQUIVALENCE

Sheldon A. Borkin

(Presented at the Fourth International Conference on Very Large Data Bases, September 1978, W. Berlin, Germany)

November, 1978

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LABORATORY FOR COMPUTER SCIENCE

CAMBRIDGE MASSACHUSETTS 02139

DATA MODEL EQUIVALENCE

Sheldon A. Borkin

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

The current proliferation of proposals for database system data models and the desire for database systems which support several different data models raise many questions concerning equivalence properties of different data models. To answer these questions, one first needs clear definitions of the concepts under discussion. This paper presents formal definitions of the terms *database, operation, operation type, application model* and *data model*.

Using this formal framework, *database state equivalence, operation equivalence, application model equivalence* and *data model equivalence* are distinguished. Three types of application and data model equivalence are defined - *isomorphic, composed operation* and *state dependent*. Possibilities for *partial equivalences* are mentioned. Implementation implications of these different equivalences are discussed.

Examples are presented using two *semantic data models*, the *semantic relation data model* and the *semantic graph data model*.

Key Words: database systems, semantic data models, data model equivalence semantic relation data model, semantic graph data model

This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-75-C-0661.

DATA MODEL EQUIVALENCE

Sheldon A. Borkin

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts

Outline

1. Introduction

The current proliferation of proposals for database system data models and the desire for database systems which support several different data models raise many questions concerning "equivalence properties" of different data models. However, a question such as "Are the relational and network data models equivalent?" can be interpreted in many different ways.

This paper presents a formal framework in which the various interpretations of such a question can be understood. Formal definitions are given to such terms as *data model* and *database*. Several different types of equivalence properties are defined and important implications of each equivalence property are discussed. The importance of a data model being *semantic* in nature is noted.

1.1 Data Models

The structure of a database as visible to the user of a database system and the operations allowed to change the database are defined in terms of a *data model*. The most often compared data models (Nijssen[26] and Date and Codd[10]) are the relational data model, as presented in Codd[6,7,8], and the DBTG[9] network model. We will assume the reader is familiar with these two data models.

However, as Nijssen[27] points out, the chosen data model provides a "mental model" of the database and "a mental model has a close analogy with religion, which is hopefully selected in freedom." The samplings of data models in Kerschberg et. al.[19] and Senko[31] mention over thirty different data models. This makes one suspect that every man will in

freedom choose his own unique religion.

1.2 Multiple Data Model Database System Architecture

The development of many different data models has both motivated and been motivated by the ANSI report[1] which recommends a database system architecture based on the concept of several levels of database descriptions. This architecture allows for multiple users accessing a shared database. The three database description levels, shown in Figure 1, include the *internal schema*, the *conceptual schema* and *external schemas*.

The internal schema specifies the types of data structures, devices and access methods which constitute the physical storage aspects of the database system. The conceptual schema

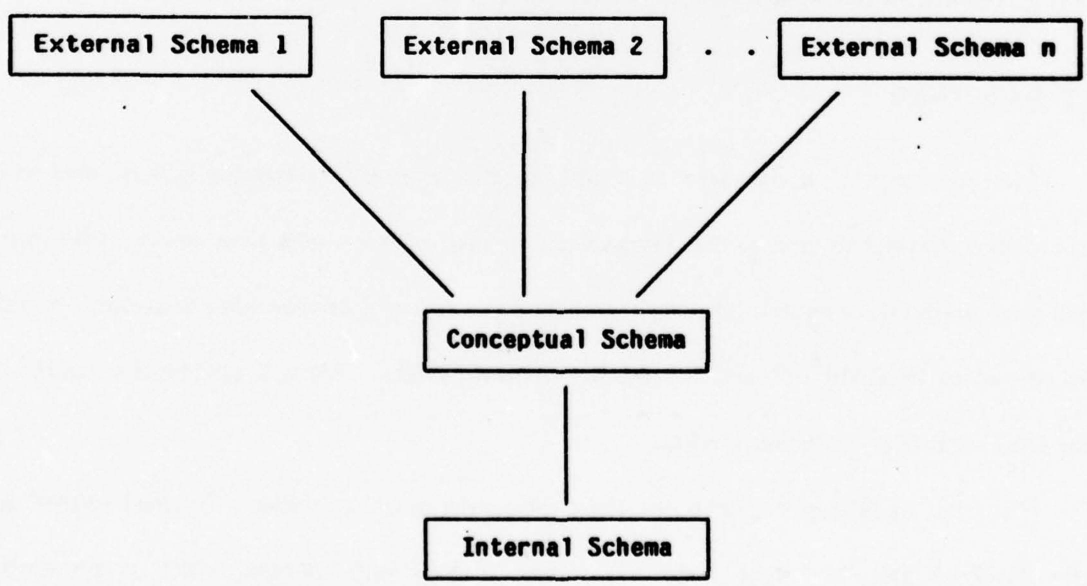


Figure 1 - ANSI Schemas

specifies the contents of the database in terms of objects significant to the application being modelled in the database. There is much discussion over the exact form which this model should take. Many suggestions are found in Nijssen[28]. Other appropriate data models will be mentioned in this paper.

The separation of the conceptual schema from the internal schema results in data independence (Date and Hopewell[11]). That is, the details of the physical storage representation can be altered without affecting the view of the database presented to the user. The centralized conceptual schema guarantees, so long as only "appropriate" external schemas are allowed, that every user sees the database and performs updates consistent with the application being modelled.

Finally, external schemas provide the database system user with a view of the database appropriate to his specific needs and *desired data model*. The schemas are to be compatible with the particular programming or query language of the user. The external schema may present to the user just a subset of the information described in the conceptual schema. While this paper will be most relevant to the case in which the external schema describes all of the data present in the conceptual schema, the definitions to be presented can be extended to handle the case where the external schema describes a subset of the conceptual schema.

This architecture requires several mapping functions - from the conceptual schema to the internal schema and from the conceptual schema to each external schema. For example, a query from a user posed in terms of an external schema must be translated into a query in terms of the conceptual schema, which must in turn be translated into a query which

actually accesses the stored data. Results of the query must likewise be translated from an internal representation to a form appropriate to the user and the external schema.

The problem of how to define the mappings between schema levels so that all users see "equivalent" models of the application for *both* retrieval and update is extremely difficult for any case other than letting the external schema be a simple subset of the conceptual schema. Klug and Tsichritzis[20] provide an example of the complexities involved.

1.3 Goals of this paper

Some questions which can be raised in light of the preceding discussion are:

- How does one know if an external schema and a conceptual schema are "equivalent"?

- How can operations in terms of the external schema be mapped to "equivalent" operations in terms of the conceptual schema?

- How does one know if two data models are "equivalent" in expressive power so that the set of expressible external schemas is equivalent to the set of expressible conceptual schemas?

To answer these questions, one first needs clear definitions of the concepts under discussion. Section 2 of this paper provides formal definitions of the terms *database*, *operation*, *operation type*, *application model* and *data model*.

Section 3 uses this formal framework to distinguish between *database state equivalence*, *operation equivalence*, *application model equivalence* and *data model equivalence*. Three types of application and data model equivalence will be defined - *isomorphic*, *composed operation* and *state dependent*. Possibilities for *partial equivalences* are mentioned.

It is claimed in Section 3 that the key definition of *database state equivalence* can be done most convincingly when both data models of concern are *semantic data models*. An example is presented.

Section 4 presents conclusions.

2. Formal Definition of Data Model

A basic premise of what follows is that the database system is used to model some portion of the "real world" which is of interest to the system user. We will call this portion of the real world the *application*. The *application state* represents a "snapshot" of the application at a given time.

The database state may consist of objects which are in a 1-1 correspondence with the application state (a common interpretation of the network approach) or it may consist of "statements" about the application state (an interpretation of the relational approach). Other approaches are possible. In any case, we assume that the database models some application which can be thought of as having a *state* and certain allowed *transitions* between states. In defining equivalences in this paper we will be concerned about different data models' capabilities in expressing these application models, their states and their transitions.

A *data model* is then defined as a (finite or infinite) set of application models.

data model = {application model_1, application model_2, . . . application model_n}

Each of the application models specifies some application or view of an application to the database system.

2.1 Definition of Application Model

An application is represented in the database system by an *application model*. To specify the allowed states and transitions for each application, an application model consists of a schema and a finite set of operation types.

application model = (schema, {operation type$_1$,operation type$_2$, . . . operation type$_n$})
The schema corresponds to the usual notion of database schema. For the relational data model it would specify the name of each relation, the domains of allowed values for each column of a relation and the integrity constraints to be satisfied by the tuples in the relations. Codd[8] calls this the "intension" of the relations. In the DBTG data model the schema would specify the various record and set types. The schema can be considered to be the "declarative" portion of the application model.

Each *operation type* is a function

operation type : (schema × arguments × database state) → database state.

Given a schema, a database state and some arguments whose types depend on the specific operation type, an operation type defines a new database state. One such possible new state is the *error* state. We will not be concerned here with retrieval operations which can be thought of as simply displaying some subset of the database state.

Operation types correspond in the relational model to *insert-tuples* and *delete-tuples*. DBTG operation types would be *store*, *delete*, *remove* and *modify*. The operation types are the "procedural" portion of the application model. Note that we consider the application model itself to be static. Clearly, at some "higher level" there must be some means of declaring and modifying the application model.

We use an application model rather than a schema and will look at application model equivalence rather than schema equivalence to emphasize the importance of allowed operations in defining a data model. Also note that the formal view here allows different application models in the same data model to have different sets of operation types. This would be necessary for data models based on the notion of abstract data types (Liskov et. al.[21] and Smith and Smith[33]).

We will distinguish between *operation types* and *operations*. An *operation* is a function which maps each possible database state into another database state.

$$\text{operation} : \text{database state} \rightarrow \text{database state}$$

Given a schema and the set of possible other arguments for each operation type, we can generate an application model's *set of allowable operations*. For example, given a relational schema, there would be an operation corresponding to the insertion or deletion of each possible set of tuples.

2.2 Definition of Database and Database State

Thus far we have used the term *database state* without any further explanation. The specific form of the database state depends on the data model under discussion. In the relational model, the state would be a mapping from relation names to sets of tuples. In Codd's[8] terminology, this is the *extension* of the relations. A DBTG state would consist of sets of records and indicators of set membership links.

Since the only database states of concern are those which can be reached by the set of allowed operations, we define the *set of valid database states* as consisting of some initial

state, most likely the "empty state", and those states consisting of the closure of the application model's set of allowable operations applied to this initial state. Each of these states is meant to represent some possible application state or else the *error* state.

Finally, we can define a *database* as specifying for an application both the current state and all possible state transitions.

database = (application model, database state)

That is, in the context of a database system there is a static component, the application model, and a dynamic component, the database state. The application model determines the set of valid operations while the database state is the database system's representation of the application state.

The definitions used to define *data model* are summarized in Figure 2.

---

data model = {application model$_1$, application model$_2$, . . . application model$_n$}

application model = (schema, {operation type$_1$, operation type$_2$, . . . operation type$_n$})

operation type : (schema $\times$ arguments $\times$ database state) $\rightarrow$ database state

operation : database state $\rightarrow$ database state

database = (application model, database state)

Figure 2 - Summary of Terms

3. Formal Equivalence Definitions

Now that the components of a data model have been defined, we can examine the various equivalences of interest. The notion of *data model equivalence* is based on *application model equivalence* which is in turn based on *database state equivalence*. We will examine these in a "bottom-up" manner.

3.1 Database State Equivalence

We would like to say that two database states expressed in terms of two different data models are equivalent if they represent identical application states. However, the precise specification of state equivalence depends on the nature of the two data models. That is, what is the *interpretation* of the database state in terms of the application?

The ease with which such interpretations can be given varies widely between data models. Such problems of interpretation led Codd[7] to the development of *normal forms* for the relational model. The significance and the limitations of these normal forms are explained by Schmid and Swenson[30] in terms of a data model with more clearly defined semantics. We will call such data models *semantic data models*. A semantic data model, rather than allowing in the database state arbitrary syntactic structures such as trees, networks or relations, attempts to provide users of the data model with a clear interpretation of the database in terms of the relevant application. Some data models which we would classify as semantic data models are Chen[5], Deheneffe et. al.[12], Hall et. al.[15] and Schmid and Swenson[30].

Most of the proposals for the ANSI *conceptual schema* are based on similar notions of

semantics (Nijssen[28]). We will call other data models, including Codd's relational model and the DBTG model, *syntactic data models*. Bachman and Daya[2] have recently proposed an extension of the DBTG model which contains notions of modelling semantics. Note that we do not claim that there is a perfectly well-defined boundary between syntactic and semantic data models. In one sense, all formally defined data models are syntactic since formal systems are in the end pure syntax. In the sense we wish to use the term, we regard a semantic data model as one which is conducive to the user having a direct interpretation of the database in terms of the application.

The difficulty of defining equivalent database states for syntactic data models can be seen by examining previous work on equivalence problems for the relational and DBTG (or similar) models. Zimmerman[35] and Fleck[14] both require that there be a relational tuple for each DBTG record plus a binary relational tuple for each DBTG set ownership-membership link. These restrictions on the form of the relational state, and hence schema, severely limit the types of information which a user might desire to appear together in a single relation.

Kay[18] allows more general relations, but allows updates to be performed only on those relations whose tuples are in a 1-1 correspondence with the DBTG records and links. The earlier works of Neuhold[25] and McGee[22] allow much more general "equivalence" mappings, but ignore the problems of performing equivalent updates. Klug and Tsichritzis[20] and Paolini and Pelagatti[29] who also allow general mappings point out the difficulties of performing equivalent updates for arbitrary conceptual to external database state mappings.

Navathe and Schkolnick[24] define a data model to facilitate "view integration". View integration is the process of developing a single model of the application consistent with each user's view of the application. This is similar to the problem of ensuring that each ANSI external schema is compatible with the conceptual schema. Navathe and Scholnick properly emphasize the importance of stating and taking into account the effects of operations altering the database state.

3.2 An Example of Equivalent Semantic Database States

The use of semantic data models can ease the task of defining equivalent database states. This section will show what it means for two database states, based on semantic data models substantially different in their means of representing *information, to be equivalent*. Borkin[3] uses the framework presented in this paper to examine the equivalence properties of two data models, the *semantic relation data model* and the *semantic graph data model*, which can be thought of as "semantic versions" of Codd's relational and the DBTG data models, respectively.

We are not attempting to present in this paper complete descriptions of these data models. Rather, we are presenting just enough details to allow the illustration of our notions of equivalence. Furthermore, it is felt that the concepts demonstrated by examining the equivalence of these data models are generally applicable to all of the other data models mentioned in this paper.

The designs of both of these semantic data models are influenced by the notion of *case grammars* (Fillmore[13] and Bruce[4]). This claims that the "underlying" meaning of natural

language sentences can be expressed as a verb phrase, a *predicate*, plus several noun phrases - one for each *case* required by the predicate. Case grammars have formed the basis for knowledge representation for use in inferential artificial intelligence systems (Hawkinson[17], Simmons[32] and Mylopoulos et. al.[23]). We propose the use of case grammars to allow the easier organization and understandability of non-inferential database systems.

3.2.1 The Semantic Relation Data Model. Figure 3 shows a semantic relation database state describing a machine shop. The state consists of three relations which are sets of *statements (tuples)*. Each relation contains the set of all true statements fitting a certain form. For example, each row in the Operate relation represents a statement of the form:

"There is a machine of type ____ with number ____ and this machine is operated by an employee named ____."

Each row in the Employees relation represents a statement of the form:

"There is an employee named ____ whose age is ____."

The first row of the Jobs relation states:

"An employee named C.Gershag is supervised by an employee named G.Wayshum and operates a machine numbered JCL181."

A *null value* is allowed so that the second row of Jobs states:

"An employee named T.Manhart has no supervisor and operates a machine numbered NZ745."

The relation Jobs contains statements specifying the agent and object cases of the predicates

be employee:object	
employee	
name	age
names	years
T.Manhart	32
C.Gershag	40
G.Wayshum	50

Employees

operate:agent	be machine:object operate:object	
employee	machine	
name	number	type
names	serial-numbers	machine-types
T.Manhart	NZ745	lathe
C.Gershag	JCL181	press

Operate

supervise:agent	supervise:object operate:agent	operate:object
employee	employee	machine
name	name	number
names	names	serial-numbers
G.Wayshum	C.Gershag	JCL181
----	T.Manhart	NZ745

Jobs

Figure 3 - Semantic Relations

supervise and operate.

The first row in a relation's heading specifies sets of *predicate : case pairs*. The second row specifies *case types*. The third row specifies *characteristics* while the fourth row specifies *domains*. The schema must contain a specification of the values comprising each domain. This data model is similar in some respects to Chen[5].

The operations allowed in the semantic relation data model are the insertion and deletion of sets of statements. In addition, the database state resulting from every successful application of one of these operations is guaranteed to satisfy a set of *constraints* specified as part of the schema.

We will not show the constraints for this example, but they could include such conditions as:

1. The names in the first column of Operate must be a subset of the names in the first column of Employees.
2. The first column of Operate may have no null values since every machine must have an operator.
3. A specific serial number may occur only once in the second column of Operate since each machine may have no more than one operator.
4. The matching of operators and machines occurring in Operate must be the same as that in Jobs.

The constraints, as fully described in Borkin[3], are semantic counterparts of functional dependencies, subset constraints and other such constraints as used in the syntactic relational model (Codd[6], Stonebraker[34] and Hammer and McLeod[16]). Some of the constraints are expressed in terms of semantic counterparts of syntactic relational algebra operations such as *projection* and *join*.

For example, to reflect the semantics of the relations, three distinct operations, *case-join*, *predicate-join* and *conjunction*, replace the syntactic *join*. Case-join combines two relations

describing different characteristics of the same predicate:case pair into a single relation. Predicate-join combines two relations describing different cases of the same predicate into a single relation. Conjunction combines two relations containing different predicates into a single relation.

3.2.2 The Semantic Graph Data Model. While the semantic relation database state contains statements about the application state, the semantic graph state is meant to consist of objects in 1-1 correspondence with the application state. This data model is similar to those of Schmid and Swenson[30] and Deheneffe et. al.[12]. Figure 4 shows a database state in the semantic graph model which is equivalent to the semantic relation state of Figure 3.

The database state consists of *entities, associations* and *characteristics* joined by *role* and *characteristic* edges. Associations correspond to the "events" in an application state which are described by some predicate. Roles correspond to the case grammar notion of cases. In Figure 4 there are two types of entities, employees and machines, and two types of associations, operation and supervision.

The operations in this data model are meant to directly model the kinds of transitions which can take place in the application. The operations allowed are the insertion or deletion of an *independent entity*, an *independent association* or a *semantic unit*. A semantic unit is a group of entities and associations which must be inserted or deleted as a single unit due to restrictions stated in the schema.

The schema corresponding to Figure 4 is shown in Figure 5. This schema states, via the distinction between solid and dotted edges, that every machine must be part of an operation association but not every employee need be in an operation association. The

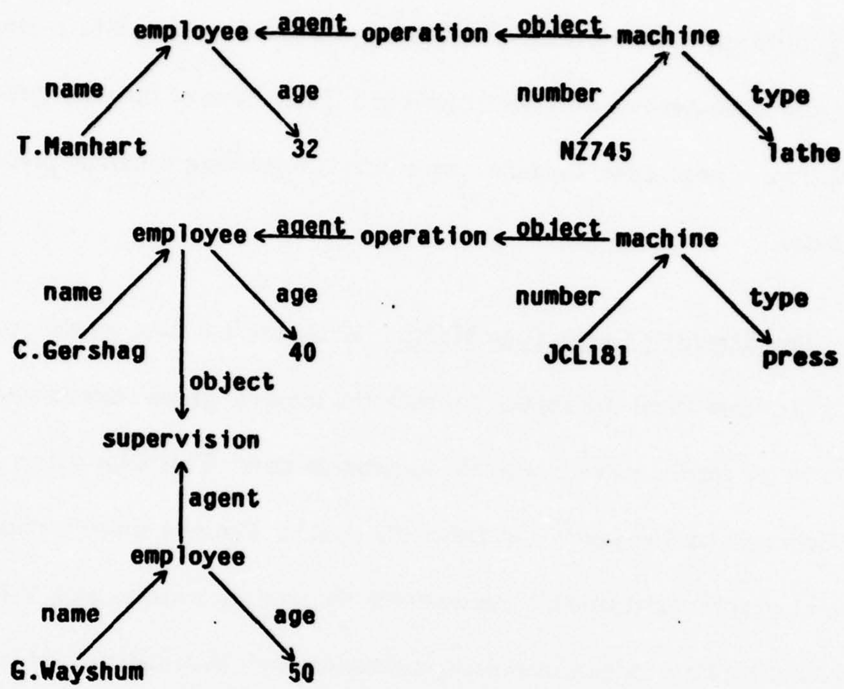


Figure 4 - Semantic Graph Database State

---

arrowheads in the schema specify functionality constraints. For example, the arrowheads state that employees are uniquely identified by their name while the identity of both the agent and object roles are necessary to uniquely identify a supervision association. They also state that a machine may belong to only one operation association. Therefore, a semantic unit is formed from a machine and its associated operation association. Whenever a machine is inserted or deleted, an operation association must also be inserted or deleted.

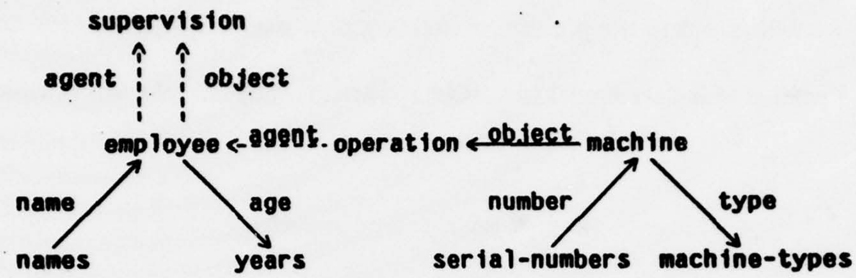


Figure 5 - Semantic Graph Schema

3.2.3 Defining Database State Equivalence. We have stated that Figures 3 and 4 represent equivalent states. We could show this by translating each relational statement into a formal logic statement and then showing that the semantic graph state is a *model*, in the formal logic sense, for the set of logical statements. A prerequisite to this would be "agreement" between the semantics of the two data models. For example, a mapping between association types and the predicate used to express information concerning each association type would be required ("supervision" and "supervise", "operation" and "operate"). That is, there must be a translation between the natural language case grammars on which the two data models are based. Hopefully, *the natural language meaning of the words would help confirm that the mappings are done correctly.*

All of the formalisms presented in this paper are also relevant for maintaining equivalence between the conceptual and internal levels. However, here it is difficult to use only semantics to define database equivalence since the internal schema presumably contains much implementation information which has no equivalent at the conceptual level. The corresponding state and operation mappings may be very complex.

3.3 Data Model and Application Model Equivalence

Now we will return to the problem of defining data model equivalence for *any* two data models. Recall that a data model is a set of application models. We will consider two data models

$$M = \{m\text{-}model_1, \ . \ . \ . \ m\text{-}model_m\}$$

and

$$N = \{n\text{-}model_1, \ . \ . \ . \ n\text{-}model_n\}.$$

Our goal is then to specify under what conditions we can say that M is equivalent to N.

3.3.1 Application Model Equivalence.

Next we want to define *application model equivalence*. That is, how to identify when application models from different data models represent the same application. For the ANSI architecture, we would want to show that the internal application model, the conceptual application model and each external application model are equivalent.

We will consider defining application model equivalence for two application models $m\text{-}model \in M$ and $n\text{-}model \in N$. We will assume that *M-ops* and *N-ops*, and *M-states* and *N-states* define the sets of allowable operations and the sets of valid states for these two application models. We will refer to database states and operations from these sets as as *m-op* and *n-op*, and *m-state* and *n-state*, respectively.

We will assume that the state equivalence correspondence has been defined between *M-states* and *N-states*. That is, given states *m-state* and *n-state*, we can tell whether or not they represent the same application state. We require that the state equivalence correspondence be *onto* both *M-states* and *N-states*. That is, the "expressive power" of the

state representation schemes are equal in both application models. It also seems reasonable to assume that for both application models, some specific application state is represented by a unique state. Hence, we require state equivalence to be a 1-1 onto correspondence. This does *not* require that there be a 1-1 mapping between *components* of equivalent states as required, for example, by the mapping of each relational tuple to either a DBTG record or a set membership link in Zimmerman[35]. The specific manner of defining the state equivalence relation is dependent upon the data model being considered.

A straight-forward approach to defining application model equivalence is to require a 1-1 onto correspondence, *operation equivalence*, between the sets of operations *M-ops* and *N-ops* such that state equivalence and operation equivalence form an isomorphism. We make the assumption that the *error* states of all application models are equivalent.

First we define *operation equivalence*.

Definition 1: A function

$$m : M\text{-}states \rightarrow M\text{-}states$$

is *operation equivalent* to a function

$$n : N\text{-}states \rightarrow N\text{-}states$$

if and only if for any two equivalent states $m\text{-}state \in M\text{-}states$ and $n\text{-}state \in N\text{-}states$, $m(m\text{-}state)$ is state equivalent to $n(n\text{-}state)$.

Then define *isomorphic application model equivalence*.

Definition 2: Two application models *m-model* and *n-model* are *isomorphically equivalent* if and only if:

i). For every operation $m\text{-}op \in M\text{-}ops$, there exists exactly one $n\text{-}op \in N\text{-}ops$ which is operation equivalent to *m-op*.

and ii). For every operation $n\text{-}op \in N\text{-}ops$, there exists exactly one $m\text{-}op \in M\text{-}ops$ which is operation equivalent to *n-op*.

This is the "most strict" form of application model equivalence we will define. However,

it may in fact be too strict. Some application models which we would like to call equivalent do not satisfy the definition. In particular, consider the semantic relation and semantic graph data models. Suppose that in the semantic relation data model we consider an operation which inserts tuples corresponding to the insertion in the graph model of several independent entities. That is, the relational operation is equivalent to the *composition* of several graph operations.

To account for this form of equivalence, we allow the definition of operation equivalence to be applied to the sets $M\text{-}op^*$ and $N\text{-}op^*$ which represent the set of all compositions of operations in the two application models. This allows the definition of *composed operation application model equivalence.*

Definition 3: Two application models *m-model* and *n-model* are *composed operation equivalent* if and only if:

i). For every operation $m\text{-}op \in M\text{-}ops$, there exists an operation $n \in N\text{-}ops^*$ which is operation equivalent to *m-op*.

and ii). For every operation $n\text{-}op \in N\text{-}ops$, there exists an operation $m \in M\text{-}ops^*$ which is operation equivalent to *n-op*.

Definition 3 does not restrict the operation equivalence correspondence to be 1-1. It does require that there be an operation equivalent to each of the simple (i.e. non-composed) operations of *M-ops* and *N-ops*. By composition of simple operations, the correspondence is *onto* $M\text{-}ops^*$ and $N\text{-}ops^*$.

In practical terms, we would hope that the operation equivalence mappings can be expressed as an algorithm rather than an explicit enumeration of an extremely large number of equivalent pairs. It is such an algorithm which would actually allow the implementation of a database system which provides users of two different data models with

access to the "same" data.

Note that based on the types of operation equivalence introduced so far, the translation of operations from one application model to an equivalent application model can be done independently of the database state. That is, operations are equivalent for all possible states. Consequently, such a translation could be done at "compile-time".

This is not necessarily always the case. The mappings of equivalent operations may be dependent upon the database state. Consider the case of adding to the graph database state of Figure 4 a supervision association between G.Wayshum and T.Manhart resulting in Figure 6. The equivalent semantic relation database state has the same Employees and Operate relations shown in Figure 3 with the new Jobs relation shown in Figure 7.

The semantic relation operation equivalent to the stated graph operation consists of the insertion of the second tuple in the Jobs relation of Figure 7. There are two important things to notice about this operation.

Firstly, we did not explicitly delete the second tuple of the Figure 3 Jobs relation. This is because the semantic relation *insert-tuples* operation type is defined to automatically delete all tuples in a relation "less than" those inserted. The partial ordering of tuples is based on all non-null domain values being greater than null ("----") and incomparable with any values other than null and itself.

More importantly, the values in the added tuple are dependent upon the database state of Figure 3. Suppose that the semantic graph state of Figure 4 had no operation association involving T.Manhart. This would not change the graph operation needed to insert the supervision association between T.Manhart and G.Wayshum.

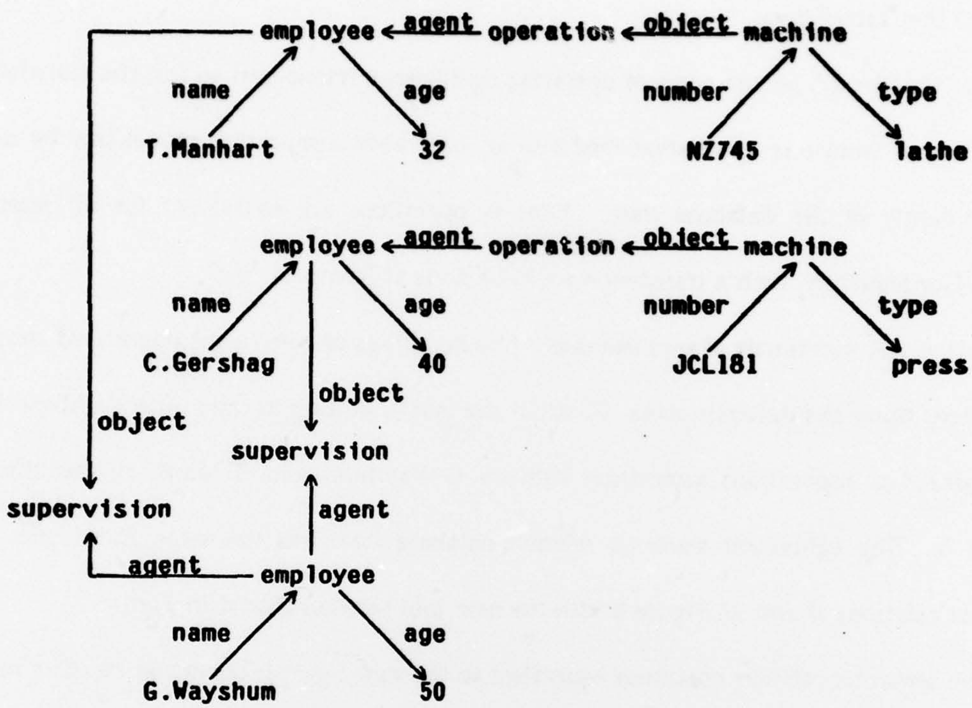


Figure 6 - Semantic Graph Database State after Insertion

In the case of the relational model, the lack of an operation association for T.Manhart would change which tuple needed to be added to reflect the new supervision. The tuple to be added would be the second tuple in Figure 8 rather than the second tuple in Figure 7. Klug and Tsichritzis[20] have also pointed out the possibility of the state affecting the definition of equivalent operations.

We reflect the possibility of state dependence in the following definitions of *state dependent operation* and *application model equivalences*:

supervise:agent	supervise:object operate:agent	operate:object	Jobs
employee	employee	machine	
name	name	number	
names	names	serial-numbers	
G.Wayshum	C.Gershag	JCL181	
G.Wayshum	T.Manhart	NZ745	

Figure 7 - Semantic Relation after Insertion

supervise:agent	supervise:object operate:agent	operate:object	Jobs
employee	employee	machine	
name	name	number	
names	names	serial-numbers	
G.Wayshum	C.Gershag	JCL181	
G.Wayshum	T.Manhart	----	

Figure 8 - Semantic Relation showing State Dependence of Insertion

Definition 4: A function

$$m : M\text{-}states \rightarrow M\text{-}states$$

and a function

$$n : N\text{-}states \rightarrow N\text{-}states$$

are *state dependent operation equivalent for a given pair of equivalent states* $m\text{-}state \in M\text{-}states$ and $n\text{-}state \in N\text{-}states$, if and only if $m(m\text{-}state)$ is state equivalent to $n(n\text{-}state)$.

Definition 5: Two application models *m-model* and *n-model* are *state dependent equivalent* if and only if for every pair of equivalent states $m\text{-}state \in M\text{-}states$ and $n\text{-}state \in N\text{-}states$,:

i). For every operation $m\text{-}op \in M\text{-}ops$, there exists an operation $n \in N\text{-}ops^*$ which is state dependent operation equivalent to *m-op* for *m-state* and *n-state*.

and ii). For every operation $n\text{-}op \in N\text{-}ops$, there exists an operation $m \in M\text{-}ops^*$ which is state dependent operation equivalent to *n-op* for *m-state* and *n-state*.

The types of application model equivalence defined are decreasingly strict. That is, isomorphic equivalence implies composed operation equivalence, and composed operation equivalence implies state dependent equivalence. There are other possibilities - we could define state dependent equivalence which does not allow composed operations. The intuition guiding the definitions presented here is that as data models become more dissimilar, they do so in several ways simultaneously. This seems justified by the examples considered.

3.3.2 Data Model Equivalence. Finally we reach the goal of defining data model equivalence.

Definition 6: Two data models M and N are *(isomorphically, composed operation, state dependent) equivalent* if and only if (isomorphic, composed operation, state dependent) application model equivalence defines a correspondence between M and N onto both M and N.

That is, if two data models are equivalent, then for any application model in one data model there is an equivalent application model in the other data model. The expressive

power of the data models are equivalent. The definition is dependent upon the type of application model equivalence required.

For the ANSI architecture, equivalence of the conceptual and external data models would mean that any conceptual application model (schema) could be viewed through the external data model, and any external application model would have a corresponding conceptual model.

The general semantic relation and the semantic graph data models are not equivalent. This is due to the great freedom in the relational model, as in some syntactic relational models, allowed in specifying constraints. As pointed out in Borkin[3], there are relational application models which do not have an equivalent graph application model. A relational application model may have either too many or too few constraints to be equivalent to a graph model. We can then say that these data models are *partially equivalent*. By restricting the allowed constraints, total state dependent equivalence can be defined for the semantic relation and graph data models.

Note that the definitions do not require that application model equivalence be 1-1. In the case of the semantic relation and graph data models, there may be several relational application models state dependent equivalent to each graph model. This corresponds to the many different ways of grouping into relations the statements corresponding to a single graph state. For example, Figure 9 (with appropriate constraints) shows a single semantic relation which is application model equivalent and state equivalent to the three semantic relations in Figure 3. Such a property makes the semantic relation model a good candidate for the ANSI external data model allowing many different relational views of a single

semantic graph conceptual application model.

4. Conclusions

A formal framework for defining data models has been presented. Using this framework, several different types of data model equivalence were defined. This framework and these equivalence definitions should be used to evaluate the suitability of data models for the different ANSI schema levels. Insight can be gained into both the data models themselves and into the type of implementation needed.

The considerations involved in actually implementing an ANSI-like architecture must include an examination of the equivalence issues discussed in this paper. Whether the

---

Machine-shop

supervise:agent	be employee:object supervise:object operate:agent		be machine:object operate:object	
employee	employee		machine	
name	name	age	number	type
names	names	years	serial-numbers	machine-types
G.Wayshum	C.Gershag	40	JCL181	press
---	T.Manhart	32	NZ745	lathe
---	G.Wayshum	50	---	---

Figure 9 - Equivalent Semantic Relation

defined equivalences are state dependent or not would have a large effect on the types of implementations allowable. The operation equivalence mappings from the external to internal levels should be analyzed as a guide for implementation efficiency considerations. The same types of equivalence mappings must be involved in the transportation of a database and associated programs from one database system to another.

Note that the framework is applicable to syntactic data models *as well as* semantic data models. We have simply pointed out that the task of comparing data models is easier when the data models of concern attempt to provide a clear interpretation of how they represent that portion of the real world which is of interest to the user.

For example, the studies in Borkin[3] defining the equivalence of the semantic relation and graph data models shed light on the syntactic relation vs. DBTG network "controversy". As in the syntactic case, the semantic relation model presents much simpler structures and operations than the graph (network) model. However, when defining a relational application model equivalent to some given graph model, it is necessary to define a large number of possibly complex constraints making updating difficult to understand.

One way or another, the complexity of the application must be represented in the application model. The choice of data models can significantly alter where the complexity is to reside. However, the ability to support equivalent relational and graph application models accessing a shared database would allow the best of both worlds - a simple relational view for retrieval and a graph model for updating.

There should be more development of semantic data models such as the semantic relation model for the external schema level. While it may be necessary to support existing

syntactic data models at the external level, all users should not be deprived of important semantic information which helps him better understand how the application is modelled. User interfaces and languages should be developed for such purposes.

References

1. ANSI/X3/SPARC Study Group on Data Base Management Systems, Interim Report, in FDT, Vol. 7, No. 2, 1975.

2. Bachman, C.W. and Daya, M., The role concept in data models, Proc. Int. Conf. on Very Large Data Bases, ACM, 1977.

3. Borkin, S.A., Equivalence Properties of Semantic Data Models for Database Systems, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, forthcoming Technical Report.

4. Bruce, B., Case systems for natural language, Artificial Intelligence 6, 1975.

5. Chen, P.P., The Entity-Relationship Model - Toward a Unified View of Data, ACM TODS, Vol. 1, No. 1, March 1976.

6. Codd, E.F., A Relational Model of Data for Large Shared Data Banks, CACM, Vol. 13, No. 6, June 1970.

7. Codd, E.F., Further Normalization of the Data Base Relational Model, Data Base Systems, Courant Computer Science Symposia Series, Vol. 6, Prentice Hall, 1972.

8. Codd, E.F., Recent Investigations in Relational Data Base Systems, Proc. IFIP Congress 1974.

9. Data Base Task Group Programming Language Committee, Codasyl, Proposal, ACM, Feb. 1973.

10. Date, C.J. and Codd, E.F., The Relational and Network Approaches: Comparison of the Application Programming Interfaces, Proc. 1974 ACM SIGMOD Workshop.

11. Date, C.J. and Hopewell, P., Storage Structure and Physical Data Independence, Proc. 1971 ACM SIGFIDET Workshop.

12. Deheneffe, C., Hennebert, H. and Paulus, W., Relational Model for a Data Base, Proc. IFIP Congress 1974.

13. Fillmore, C., The case for case, in Universals in Linguistic Theory, Bach and Harms, eds., Holt, Rinehart and Winston, 1968.

14. Fleck, M., On the Equivalence of Data Base Models, Technical Report TR 25.144, IBM Laboratory Vienna, July 1975.

15. Hall, P., Owlett, J. and Todd, S., Relations and Entities, in Modelling in Data Base Management Systems, Nijssen, G.M., ed., North-Holland Publishing Company, New York, 1976.

16. Hammer, M.M. and McLeod, D.J., Semantic Integrity in a Relational Data Base System, Proc. Int. Conf. on Very Large Data Bases, ACM, Sept. 1975.

17. Hawkinson, L., The Representation of Concepts in OWL, Proc. IJCAI 1975.

18. Kay, M.H., An assessment of the Codasyl DDL for use with a relational subschema, in Data Base Description, Douque, B.C.M. and Nijssen, G.M., eds., North-Holland Publishing Company, New York, 1975.

19. Kerschberg, L., Klug, A. and Tsichritzis, D., A Taxonomy of Data Models, in Systems for Large Data Bases, Lockemann, P.C. and Neuhold, E.J., eds., North-Holland Publishing Company, New York, 1977.

20. Klug, A. and Tsichritzis, D., Multiple view support within the ANSI/SPARC framework, Proc. Int. Conf. on Very Large Data Bases, ACM, 1977.

21. Liskov, B., Snyder, A., Atkinson, R. and Schaffert, C., Abstraction Mechanisms in CLU, CACM, Vol. 20, No. 8, August 1977.

22. McGee, W.C., A Contribution to the Study of Data Equivalence, in Data Base Management, Klimbie, J.W. and Koffeman, K.L., eds., North-Holland Publishing Company, New York, 1974.

23. Mylopoulos, J., et. al., TORUS - A Natural Language Understanding System for Data Management, Proc. IJCAI 1975.

24. Navathe, S.B. and Schkolnick, M., View representation in logical data base design, Proc. ACM SIGMOD 1978.

25. Neuhold, E.J., Data Mapping: A Formal Hierarchical and Relational View, Bericht 10, Universitat Karlsruhe, Institut Fur Angewandte Informatik, Feb. 1973.

26. Nijssen, G.M., Data structuring in DDL and the relational data model, in Data Base Management, Klimbie, J.W. and Koffeman, K.L., eds., North-Holland Publishing Company, New York, 1974.

27. Nijssen, G.M., A Gross Architecture for the next generation Database Management Systems, in Modelling in Data Base Management Systems, Nijssen, G.M., ed., North-Holland Publishing Company, New York, 1976.

28. Nijssen, G.M. ed., Modelling in Data Base Management Systems, North-Holland Publishing Company, New York, 1976.

29. Paolini, P. and Pelagatti, G., Formal definitions of mappings in a data base, Proc. ACM SIGMOD 1977.

30. Schmid, H.A. and Swenson, J.R., On the Semantics of the Relational Data Model, Proc. ACM SIGMOD 1975.

31. Senko, M.E., Data structures and data accessing in data base systems past, present, future, IBM Systems Journal 16, No. 3, 1977.

32. Simmons, R.F., Semantic Networks: Their Computation and Uses for Understanding English Sentences, in Computer Models of Thought and Language, Schank, R.C. and Colby, K.M., eds., W.H. Freeman and Co., San Francisco, 1973.

33. Smith, J.M. and Smith, D.C.P., Database Abstractions: Aggregation, CACM, Vol. 20, No. 6, June 1977.

34. Stonebraker, M., High Level Integrity Assurance in Relational Data Base Management Systems, Memorandum ERL-M473, Electronics Research Laboratory, Univ. of Cal., Berkeley, August 1974.

35. Zimmerman, K., Different Views of a Data Base: Coexistence Between Network Model and Relational Model, Technical Report TR 25.143, IBM Laboratory Vienna, June 1975.

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office/Boston
495 Summer Street
Boston, MA 02210
1 copy

Office of Naval Research
Branch Office/Chicago
536 South Clark Street
Chicago, IL 60605
1 copy

Office of Naval Research
Branch Office/Pasadena
1030 East Green Street
Pasadena, CA 91106
1 copy

New York Area Office
715 Broadway - 5th floor
New York, N. Y. 10003
1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375
6 copies

Assistant Chief for Technology
Office of Naval Research
Code 200
Arlington, VA 22217
1 copy

Office of Naval Research
Code 455
Arlington, VA 22217
1 copy

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380
1 copy

Office of Naval Research
Code 458
Arlington, VA 22217
1 copy

Naval Electronics Lab Center
Advanced Software Technology
Division - Code 5200
San Diego, CA 92152
1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Math Department
Bethesda, MD 20084
1 copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
(OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Mr. Kin B. Thompson
Technical Director
Information Systems Division
(OP-91T)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York, N. Y. 09501
1 copy